ЦИФРОВАЯ
ИНДУСТРИАЛЬНАЯ
ПЛАТФОРМА

Руководство для системы развертывания (zifctl) (2.16.0)

Zyfra Industrial Internet of Things Platform (ZIIoT)

Изменения в документе

Версия	Дата	Автор	Описание
1.0	01.12.2023	Пеплин Ф.Н.	Создание документа
1.1	16.01.2024	Пухов А. В.	В разделе 4 изменена версия keycloak по умолчанию с 17 на 21.

Содержание

1. Развертывание Платформы в виде docker-образа 5
1.1. Логин в docker registry 5
1.2. Получение образа и запуск контейнера 5
1.3. Получение скрипта-обертки (wrapper script) 6
1.4. Создание новой конфигурации окружения (если развертывается новый экземпляр Платформы) 7
1.5. Создание ключа шифрования и задание переменной ZIF_AGE_KEY 7
1.6. Создание новой конфигурации окружения 8
1.7. Доступ к кластеру Kubernetes или OpenShift для развертывания и переменные ZIF_SERVER и ZIF_TOKEN 9
1.8. Развертывание или обновление Платформы 10
1.9. Удаление развернутой Платформы из кластера 10
1.10. Отладочные команды 11
1.11. Получение списка требуемых docker-образов для развертывания и export/import образов 12
2. Руководство по обновлению 13
3. Генерация документа (zifctl doc) env-values - конфигурирования переменных 14
4. Изменения Apache Keycloak, начиная с релиза Платформы 2.16.0 15
5. Профили postgres (postgres profiles) или настройки postgres для мультикластерных PG-инсталяций 17
5.1. Переключение сервиса Keycloak на другой PG 17
5.2. Переключение сервиса X (не keycloak) на другой PG 18
6. Автоматическое создание и конфигурация топиков Kafka в процессе развертывания 22
7. Включение TLS, аутентификации и авторизации в Redis 26
7.1. Включение режима TLS (Transport Layer Security) 26
7.2. Включение режима AUTH (аутентификация) 26
7.3. Включение режима AUTH и ACL 27
7.4. Включение режима TLS, AUTH и ACL 27
7.5. Режим публикации внешнего Ingress/Route соединения для сервера Redis 28
8. Включение отдельного Redis для инфраструктуры и настройка Apache Nifi 29
8.1. Установка Redis для инфраструктуры 29
8.2. Конфигурация Apache Nifi 31
8.3. Конфигурация сервисов контроллера RedisConnectionPoolService и RedisDistributedMapCacheClientService 32
8.4. Включение сервисов контроллера RedisConnectionPoolService и RedisDistributedMapCacheClientService 33
8.5. Конфигурация процессоров GenerateFlowFile и PutDistributedMapCache 36
8.6. Запуск процессоров GenerateFlowFile и PutDistributedMapCache 38
8.7. Тестирование Redis 38

9. Требования к рабочей станции, с которой будет производиться развертывание, и ее окружению 39

1. Развертывание Платформы в виде docker-образа

Внимание! Данный раздел применим только для Релиза 2.9.0 и более поздних.

Система развертывания **Платформы** (обычно называемая **zifctl** по названию скрипта запуска) до релиза 2.9.0 поставлялась в виде исходного кода, как **git**-репозиторий. Для установки требовалось скачать репозиторий и запустить из него скрипт **zifctl** с необходимыми параметрами.

Начиная с релиза 2.9.0, инструмент развертывания **Платформы** поставляется в виде готового **docker**-образа, тег которого совпадает с версией устанавливаемой версии **Платформы** (плюс, возможно, суффикс-идентификатор сборки).

Это дает следующие преимущества:

1) Нет необходимости иметь доступ к **git**-репозиторию на целевой площадке или приносить его туда в архиве. Достаточно только наличия **docker**-образа в локальном **registry** (образы для Платформы все равно туда переносить или зеркалировать).
2) Нет необходимости в скачивании и переносе **helm-chart** для развертывания Платформы. Необходимые версии чартов положены в образ на этапе сборки.

Из недостатков можно отметить более сложную схему запуска инструмента т.к. приходится пользоваться **docker**-контейнером, в который необходимо передавать пути с локальной машины. Для облегчения этой задачи разработан специальный скрипт-обертка.

Для установки Платформы необходимо иметь доступ к **docker-registry**, где размещаются образы самой Платформы, ее инфраструктурных зависимостей и контейнер с системой развертывания.

Далее в командах будет использоваться репозиторий **Цифры**: https://registry.dp.zyfra.com/repository/. Если Платформа устанавливается в закрытом контуре, следует заменить адреса **registry**, и путь к контейнеру (для переноса контейнеров в registry в закрытом контуре, можно воспользоваться новой командой **zictl image scripts**).

1.1. Логин в docker registry

Для **Qauy** необходимо использовать **encrypted** `password` для логина через **CLI**, который можно получить в **UI** в своем профиле (**Account Settings**) и разделе **Docker CLI Password**:

```
echo $PASSWORD | docker login registry.dp.zyfra.com -u $USERNAME --password-stdin
```

1.2. Получение образа и запуск контейнера

Пробный запуск и получение помощи по доступным командам, получение информации о версии Платформы, которую развертывает контейнер, выглядит следующим образом:

```
# Получить образ
docker pull registry.dp.zyfra.com/infra/zifctl:2.9.0

# Запуск простых команд сразу с образа
docker run --rm  registry.dp.zyfra.com/infra/zifctl:2.9.0 --help
```

```
docker run --rm  registry.dp.zyfra.com/infra/zifctl:2.9.0 version
```

1.3. Получение скрипта-обертки (wrapper script)

Для интерактивной работы в терминале рекомендуется получить из контейнера скрипт-обертку для упрощенного запуска (скрипт позволяет указывать локальные пути и передает переменные окружения **ZIF_SERVER**, **ZIF_TOKEN**, **ZIF_AGE_KEY**, **ZIF_ENV_PATH** в запускаемый контейнер).

С точки зрения запуска скрипт-обертка примерно аналогична **zifctl** в прошлых релизах. Использование этого скрипта не обязательно (можно использовать **docker run**), но он добавляет возможность записи вывода контейнера в лог-файл (сам контейнер выводит все сообщения просто на консоль).

Скрипт-обертка обновляется вместе с системой развертывания. Необходимо обновлять его при переходе на новые версии системы и Платформы (каждый новый **docker**-образ может содержать новую версию **wrapper**).

Внутри скрипта зашита версия контейнера, с которого он получен и по умолчанию он будет запускать команды именно на нем. Посмотреть какой контейнер будет запускать скрипт можно выполнив команду **zifctl version.**

Для запуска этого скрипта на машине должен быть установлен **Python 3** (достаточно только штатных библиотек).

```
# Опционально скопировать скрипт в путь доступный через PATH
docker run --rm  registry.dp.zyfra.com/infra/zifctl:2.9.0 get-wrapper > zifctl
chmod +x zifctl
sudo cp zifctl /usr/local/bin
```

Проверка работы скрипта-обертки:

```
# Помощь по командам
./zifctl --help

# Информация о версиях используемых образов в скрипте, информация о версии Платформы
./zifctl version
```

Внимание! Скрипт-обертка обновляется вместе с системой развертывания. Необходимо обновлять его при переходе на новые версии системы и Платформы (каждый новый **docker**-образ может содержать новую версию **wrapper**).

Внимание! Внутри скрипта зашита версия контейнера, с которого он получен и по умолчанию он будет запускать команды именно на нем. Посмотреть какой контейнер будет запускать скрипт можно выполнив команду **zifctl version**

Внимание! Также внутри скрипта зашита ссылка на реестр/имя контейнера. Это можно поменять при помощи аргумента **--zifctl-registry** docker.idp.yc.ziiot.ru/infra (например, для переключения на реестр ЦИП).

1.4. Создание новой конфигурации окружения (если развертывается новый экземпляр Платформы)

Если **Платформа** устанавливается с нуля, то нужно создать конфигурацию окружения (конфигурации экземпляра) Платформы.

Конфигурация окружения представляет собой каталог с файлами, в которых сохранены все необходимые данные для разворачивания с нуля экземпляра **Платформы** или его обновления.

В общем случае предполагается, что конфигурация хранится в **git** и платформа ставится и обновляется по модели **GitOps**, в которой «источником правды» служит репозиторий, и по данным из него создаются все объекты в кластере (кроме **PVC** с данными, которые сохраняются при обновлениях).

1.5. Создание ключа шифрования и задание переменной ZIF_AGE_KEY

В конфигурации окружения так же хранятся и все пароли, **connection string** и прочие секреты, необходимые для работы Платформы. Т.к. секреты хранятся вместе с конфигурацией в **git**, они должны быть зашифрованы.

Система развертывания обеспечивает прозрачное шифрование секретов с помощью **Mozilla SOPS** (https://github.com/mozilla/sops) и утилиты (бэкенда) **age** (https://github.com/FiloSottile/age).

Шифрование требует секретного ключа (**private key**) для утилиты **age**, обычно уникального для данного экземпляра.

При развертывании нового экземпляра вам необходимо создать такой ключ и сохранить его (ключ не должен сохранятся в **git**-репозитории вместе с данными, которые он шифрует). Команда для создания нового ключа и сохранения его в файл:

```
# Создание нового ключа шифрования и запись его файл
./zifctl secrets new-age-key > key.txt
```

Полученный файл будет иметь вид ниже. Строка, начинающаяся с **AGE-SECRET-KEY-...** и есть нужный закрытый ключ. Полученный файл необходимо сохранить любым надежным способом:

```
AGE-SECRET-KEY-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Полученный ключ требуется указывать практически для всех команд утилиты развертывания, которые выполняют любую работу с экземпляром платформы. Для упрощения работы можно записать этот ключ в переменную окружения **ZIF_AGE_KEY**, и не указывать параметр **--age-key** во всех командах ниже (при постоянной работе с одной площадкой можно записать создание этой переменной в профиль **shell**):

```
export                                                    ZIF_AGE_KEY="AGE-SECRET-KEY-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

1.6. Создание новой конфигурации окружения

Новая конфигурация окружения создается командой **init**. Папка для создания задается параметром **--env-path** (папка должна существовать). В этой папке создаются файлы **env-values.yaml** и **env-secrets.yaml** и несколько дополнительных.

Все параметры команды **init** кроме **--env-path** *и* **--age-key** *(*или переменная **ZIF_AGE_KEY***)* опциональные и фактически задают значения для основных параметров в файлах **env-values.yaml** и **env-secrets.yaml**. Их можно задать или изменить и после создания каталога с конфигурацией (т.к. секреты в **env-secrets.yaml** зашифрованы то задать логин и пароль **registry** будет сложнее чем остальные).

```
# Помощь по параметрам команды init
./zifctl init --help


# Информация о версиях используемых образов в скрипте, информация о версии Платформы
./zifctl init --env-path /path/to/env/config --age-key $KEY \
      --namespace $NAMESPACE \
      --sa $SA \
      --hostname $ZIIOT_HOSTNAME \
      --storage $STOR_CLASS \
      --registry $REGISTRY \
      --registry-username $REG_USERNAME \
      --registry-password $REG_PASSWORD \
      --verbose
```

Основные параметры команды **init**, задающие значения в новой конфигурации:

- **--namespace** — задает **Namespace** куда будет устанавливаться **Платформы**.
- **--sa** — **Service Account** для запуска инфраструктурных контейнеров, которые требуют запуска от конкретного **UID** (т.е. повышенных привилегий). Актуально в первую очередь для **OpenShift/OKD** инсталляций. Если не задан, то все запускается от учетной записи по умолчанию.
- **--hostname** — базовое доменное имя для доступа к экземпляру Платформы (по этому имени будет доступен портал, а сервисы будут иметь пути вроде **https://<base-hostname>/<service-name>** *(указывается не ссылка* **URL***, а именно* **hostname***).*
- **--storage** — имя **Storage Class** для создания **PVC** всей инфраструктурой. Если не задано, то используется **SC** по умолчанию в кластере.

- **--registry** — имя репозитория откуда кластер **K8s/OpenShift** должен брать контейнеры Платформы и инфраструктуры (по умолчанию **registry.dp.zyfra.com**).
- **--registry-username** — имя пользователя для репозитория. Из него и **registry-password** формируется **pullSecret**, в большинстве случаев должно быть указано, если только это не полностью открытый внутренний репозиторий).
- **--registry-password** — пароль пользователя для репозитория.

При необходимости внесите изменения в созданную в команде **init** конфигурацию окружения. Для создания простого стенда Платформы с размещением всей инфраструктуры в кластере достаточно указанных выше параметров.

Все основные параметры развертывания указываются в **env-values.yaml.**

Секреты для инфраструктурных сервисов указываются в файле **env-secrets.yaml.** Данные в этом файле зашифрованы с помощью `age` и для доступа к ним можно воспользоваться группой команд **zifctl secrets** (краткая помощь **zifctl secrets --help**).

Секреты генерируются автоматически, и вам нужно их править только если используется внешние инфраструктурные сервисы, например СУБД, которые развернуты независимо от **zifct** (или если не указан пароль к **registry** при выполнении **Init**).

1.7. Доступ к кластеру Kubernetes или OpenShift для развертывания и переменные ZIF_SERVER и ZIF_TOKEN

Для развертывания Платформы в кластер **Kubernetes** или **OpenShift** вам понадобится токен сервисного аккаунта или пользователя, позволяющий выполнять изменения в указанном **Namespace** (у него должна быть роль **admin** или близкая к этому на данный **namespace**).

- Если Платформа устанавливается в **Kubernetes**, то необходимо создать **Service Account** для этой задачи и получить его токен (см. о **SA** и токенах по ссылке, токен указывается в раскодированном виде, не в **base64**).
- Если Платформа устанавливается в **OpenShift**/**OKD** то есть возможность получить токен через **UI** для своей рабочей записи, или создать **SA** для установки так же как и для **Kubernetes**.
- Этот **Service Account** не обязан быть тем же, что и указанный для запуска подов в **env-values.yaml**. Он может отличаться, т.к. обычно аккаунту для запуска подов не рекомендуется давать права для внесения изменений в кластер.

Для всех команд утилиты развертывания, которые требуют доступа в кластер **Kubernetes/OpenShift**, необходимо задавать два параметра командной строки **--server** и **--token**, в которых указывается **URL API** кластера и токен для доступа соответственно.

При работе с одной и той же площадкой эти значения можно задать в виде переменных окружения и не указывать их в каждой команде (так же как ключом шифрования и переменной **ZIF_AGE_KEY**) или прописать их в профиль **shell**:

```
export ZIF_SERVER="https://api.cluster.local"
export ZIF_TOKEN="eyJhbGcNYWNjb3VuIsImtpZCI6IkZLOE5DSUlPZ0dtMXhPYUhvTDc2eGtf..."
```

1.8. Развертывание или обновление Платформы

Команда для развертывания и для обновления платформы одна и та же. Практически все операции, выполняемые системой развертывания, идемпотентны и при повторном выполнении установки, поверх уже выполненной ранее ничего не должно измениться.

Команда установки («синхронизации» конфигурации с конфигурацией в кластере). Параметры **--age-key**, **--server**, **--token** могут быть заданы через переменные окружения **ZIF_AGE_KEY**, **ZIF_SERVER** и **ZIF_TOKEN** соответственно).

```
# Запуск развертывания с выводом информации только на консоль
./zifctl sync --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose


# Запуск развертывания с выводом информации на консоль и в лог-файл с ротацией (параметр --log-keep не обязательный, по умолчанию сохраняется 10 последних логов)
./zifctl sync --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose --log-path /var/logs/ziiot-deploy --log-keep 20
```

Если выполняется установка поверх предыдущей версии, то в общем случае должно выполниться обновление сервисов до новой версии.

Но т.к. **Платформа** представляет собой сложный комплекс из большого количества сервисов и компонентов, то полностью автоматическое обновление условно гарантируется только в рамках одного релиза (например, из 2.9.0 в 2.9.1), а обновление между релизами (например, из 2.8.3 в 2.9.0) может потребовать ручных операций.

Перед обновлением необходимо прочитать **Release Notes Платформы** и системы развертывания и убедиться, что нет обязательных ручных операций.

1.9. Удаление развернутой Платформы из кластера

Команды для удаления развернутой платформы из кластера:

```
# Удаление всех сервисов и инфраструктуры без удаления PVC
./zifctl delete --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose


# Удаление всех сервисов и инфраструктуры с удалением PVC (полная очистка). Созданные в Namespace дополнительно вручную объекты удалены не будут.
./zifctl clean --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose
```

1.10. Отладочные команды

При выявлении каких-то проблем с развертыванием платформы на любом этапе работы системы, можно воспользоваться командами **write-values** и **template,** а также ключом выдачи отладочной информации **--verbose** и **–debug.**

Система развертывания **Платформы** в целом работает по следующей схеме (схема упрощенная, в ней не указаны различные вспомогательные скрипты на разных этапах и не указана система инициализации сервисов, запускаемая уже в кластере после загрузки манифестов):

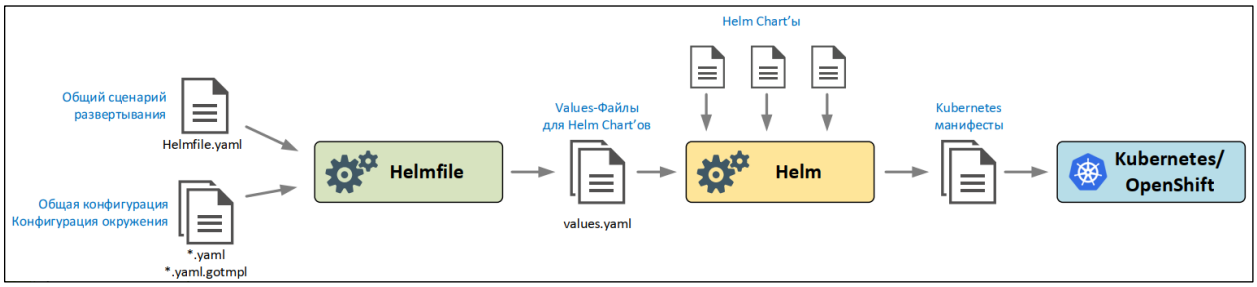

Рисунок 1.1 Отладочные команды

1) На основании файлов сценариев **Helmfile***, шаблонов конфигурации (**values***) и при участии доп. скриптов создаются параметры для **Helm Chart** (эту функцию выполняет утилита **Helmfile**).
2) Полученные параметры (**values**) передаются в **Helm Chart**, из которых генерируются манифесты для объектов **K8s** (эту функцию выполняет **Helm**).
3) Созданные манифесты загружаются в кластер **Kubernetes**, при необходимости с обновлением уже существующих (эту задачу тоже выполняет **Helm**).
4) Запускаются объекты заданий (**Job**) внутри кластера, которые выполняют начальную инициализацию развернутой инфраструктуры и сервисов (этот этап на схеме не указан).

В системе развертывания предусмотрены команды, позволяющие посмотреть генерируемые значения для **chart** и генерируемые манифесты на основании этих значений (аналогично командам **Helm**):

```
# Запись значений, полученных на первом этапе, которые затем должны быть переданы в Helm Chart. Значения записываются по пути, указанном в параметре --output-path

./zifctl write-values --env-path /path/to/env/config --age-key $KEY --output-path $OUTPUT_PATH


# Запись манифестов, сгенерированных Helm на основании переданных значений (результат второго этапа). Манифесты записываются по пути указанном в параметре --output-path

./zifctl clean --env-path /path/to/env/config --age-key $KEY --server $SERVER --token $TOKEN --verbose
```

Количество выдаваемой на консоль (и в лог при использовании параметра **--log-path**) информации регулируется двумя опциональными параметрами **zifctl**:

- **--verbose —** включает отладочные сообщения из скриптов самой системы развертывания **zifctl** и вызываемых им.

- **--debug —** включает помимо отладочных сообщений системы развертывания еще и отладочные сообщения в **Helmfile** и **Helm** (а также в скрипте обертке).

1.11. Получение списка требуемых docker-образов для развертывания и export/import образов

Для запуска Платформы в кластере **Kubernetes** требуются **docker**-образы сервисов Платформы, образы всей устанавливаемой инфраструктуры (**Postgres**, **Cassandra**, **Kafka** и т.д.) и образ самой системы развертывания (**zifctl**).

В отличие от предыдущих релизов, образы с **Helm Chart** и копия **git**-репозитория не требуются (это все включено в образ **zifctl**).

Все необходимые **docker**-образы размещаются в репозитории компании **Zyfra**, который система развертывания использует по умолчанию: https://registry.dp.zyfra.com.

Платформу часто приходится развертывать в закрытых контурах, куда **docker**-образы приходится переносить вспомогательными средствами, поэтому в систему развертывания включены команды, позволяющие получить список требуемых образов и сгенерировать заготовки скриптов для их выгрузки и загрузки в другой репозиторий.

```
# Получение списка требуемых docker-образов. Формат вывода указывается в параметре --output и может быть: table, list, json, yaml, csv  (для удобства интеграции с инструментами для экспорта-импорта образов)
./zifctl image list --output table

# Получение заготовок скриптов для экспорта (pull-save) и импорта в другой репозиторий (load-tag-save). Скрипты сохраняются по пути --output-path
./zifctl image scripts --output-path /path/to/save/scripts

# Получение заготовок скриптов для экспорта (pull-save) и импорта в другой репозиторий (load-tag-save). Скрипты сохраняются по пути --output-path
./zifctl image scripts --output-path /path/to/save/scripts
```

2. Руководство по обновлению

Специальные требования к процессу обновления:

1) Для некоторых топиков `debezium` заданы параметры в `services/kafka-topics-list.yaml`.

   При обновлении с предыдущего релиза и нестандартной конфигурации статичных топиков `debezium`, необходимо их отразить в `services/service-list-patch.yaml`. Количество партиций в топиках нельзя уменьшать автоматически. Управление доступно только для топиков начинающих с `<tenantid>__`.

2) Удален параметр `keylocak.baseUri`, при обновлении с предыдущего релиза его необходимо удалить из `env-values.yaml`.

3. Генерация документа (zifctl doc) env-values - конфигурирования переменных

Добавлена команда `zifctl doc`, вместе с подкомандой (пока единственной) `env-values`, которая и генерирует документацию по env-values.yaml.

Обязательный параметр: `--ouput=path` - папка в которую сохранять документацию. Необязательный: `--format` - принимает значения `md/html`, по умолчанию md

Пример команды:

1) Запуск из `docker`-образа:

```
docker run --rm -it \
-v "<путь для сгенерированных файлов документации>":/output-path \
docker-group.idp.yc.ziiot.ru/infra/zifctl:2.16.0 \
doc env-values --output-path=/output-path
```

По указанному пути будет сгенерирован файл `env-values.md` в формате `markdown`.

По указанному пути будет сгенерирован файл `env-values.html` в формате `html`:

```
docker run --rm -it \
-v "<путь для сгенерированных файлов документации>":/output-path \
docker-group.idp.yc.ziiot.ru/infra/zifctl:2.16.0 \
doc env-values --output-path=/output-path --format html
```

2) Запуск из враппера (wrapper) zifctl:

```
docker pull docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0
docker run --rm  docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0 get-wrapper > zifctl
./zifctl doc env-values --output-path .
```

По указанному пути будет сгенерирован файл `env-values.md` в формате `markdown`:

```
docker pull docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0
docker run --rm  docker.idp.yc.ziiot.ru/infra/zifctl:2.16.0 get-wrapper > zifctl
./zifctl doc env-values --output-path . --format html
```

По указанному пути будет сгенерирован файл `env-values.html` в формате `html`.

4. Изменения Apache Keycloak, начиная с релиза Платформы 2.16.0

Обновление текущей версии 17, входящей в состав Платформы на версию 21.

Обновление не должно повлиять или исказить/повредить данные как самой Платформы, так и информацию о текущих пользователях.

Изменения:

1) `baseUri` больше не будет использоваться, поэтому был удалён из модели конфига `keycloak`.
2) `KEYCLOAK_URL` не используется и была удалена.
3) `Helm chart keycloak` в будущем будет удален так как будет использоваться `helm`-чарт от `Bitnami`.
4) Добавлен новый класс `version` для указания версии `keycloak`, которую необходимо развернуть.
5) Изменена логика установки значения `default_servicename` — для версии 17 имя сервиса было с добавлением `-http`.
6) Добавлен новый файл с переменными `zif-keycloak21.yaml.gotmpl` для развертывания нового чарта `keycloak`.
7) Внесены изменения в чарте `zif-infra-ingress` чтобы при изменении версии ссылка `/auth` ссылалась на верный `keycloak` сервис (`zif-keycloak-http` или `zif-keycloak21`).
8) Внесены изменения в главный `helmfile` чтобы при установке выбиралась версия, указанная в конфигурационном файле `env-values.yaml`.
9) В базовый образ добавлен `keycloak-config-cli-21.0.1` для работы с серверной версией 21 `keycloak`.
10) Образ `zif-keycloak21` содержит темы `zif`. Темы необходимо было переработать для работы с новой версией.
11) Образ `zif-keycloak21` подготовлен для работы в `ОКД` без использования `security context`.

Необходимые действия для обновления:

Внимание! Необходимо удалить `baseUri` из блока `keycloak` в конфигурационном файле `env-values.yaml`:

1) Откройте файл `env-values.yaml` на редактирование и удалите из блока `keycloak` строку:

```
baseUri: '[http://zif-keycloak-http.dev-guseynov/auth'](#описание-задачи)**
```

2) Укажите версию 21, если хотите обновить текущий `keycloak`:

```
version: 21
```

Если версия не указана, то по умолчанию будет взята версия 21.

Файл до обновления:

```
env-config.yaml
## Конфигурация встроенного keycloak
keycloak:
  installed: True
  baseUri: 'http://zif-keycloak-http.dev-guseynov/auth'
  baseExtUri: 'https://dev-guseynov.kube07.yc.ziiot.ru/auth'
```

Файл после обновления:

```
env-config.yaml
## Конфигурация встроенного keycloak
keycloak:
  installed: True
  baseExtUri: 'https://dev-guseynov.kube07.yc.ziiot.ru/auth'
  version: 21
```

5. Профили postgres (postgres profiles) или настройки postgres для мультикластерных PG-инсталяций

Малые инсталяции платформы или инсталяции с небольшим количеством данных позволяют обойтись скейлингом Платформы в `kubernetes`.

В редких случаях требуется не только выносить `PG` на ВМ (для удовлетворения требований `ИБ`), но и раскидывать `БД` инсталяции по разным кластерам `PG` для разгрузки ВМ и удовлетворения нужного количествава "9-ок" по доступности.

Начиная с версии `zifctl 2.16.X` появилась возможность в конфигурации учитывать такое разделение. Как его настроить описано ниже:

Внутренняя структура работы с `PG` у Платформы:

К PG ходят 2 сущности платформы:

- `init job`, которые получают `user/pass` как `admin user`, `dbname` генерируют на лету на основе `tenantID` и соответствующего сервиса, `host/port/connectdb` из переменных `ENV (POSTGRES_HOST / POSTGRES_PORT / POSTGRES_CONNECT_DB)`.
- сервисы Платформы, который получают `user/pass/dbname/host/port` из переменных окружения.

Начиная с версии `zifctl 2.16.X` можно полноценно переместить сервис на другие `postgres.host / postgres.port`, а `init job` заставить создать БД/поправить права на неё и т.д. на нужном вам кластере с помощью `postgres profiles` и параметра `connectDB`.

Внимание! Для сервиса настройка `connectDB` не нужна. Эта настройка нужно только для `init job` для выполнения задач по созданию БД сервиса, пользователя и создания/исправления прав на БД. Т.е. чтобы выполнить запрос `CREATE DATABASE` и подобные `init job` должна куда-то подключится (не указывая БД при запуске командочке в `psql` вы на самом деле подключаетесь к `БД postgres`).

5.1. Переключение сервиса Keycloak на другой PG

В рамках `Keycloak` теперь возможно полноценно с текущей версии начинается работать на двухкластерной инсталяции Платформы, когда всё кроме `keycloak` работало на `default.cluster`, а сам `keycloak` на `new.cluster`:

- создайте новый профиль `PG` и переопределить в нём `host/port/connectDB` сущности (одну из них как минимум):

```
$ grep -A 11 -F 'postgres:' ./env-config/env-values.yaml
postgres:
  installed: True
  host: 'default.cluster'
  port: 5432
  connectDB: postgres
  storageSizeMB: 5120
```

```
  profiles:
    fakeprofilename:
      host: 'new.cluster'
      port: 5432
      connectDB: postgres
```

- переключить сервис `keycloak` на нужный профиль в `env-values.yaml`:

```
$ grep -A 4 -F 'keycloak:' ./env-config/env-values.yaml
keycloak:
  installed: True
  postgresProfile: fakeprofilename
  baseUri: 'http://zif-keycloak-http.dev-dyukov/auth'
  baseExtUri: 'https://dev-dyukov.kube07.yc.ziiot.ru/auth'
```

5.2. Переключение сервиса X (не keycloak) на другой PG

В рамках `Keycloak` теперь возможно полноценно с текущей версии начинается работать на двухкластерной инсталяции Платформы, когда всё кроме `keycloak` работало на `default.cluster`, а сам `keycloak` на `new.cluster`:

В текущей реализации Платформы сервис `X` можно полноценно подключить к работе на двухкластерной инсталяции Платформы, когда всё кроме `X` работало на `default.cluster`, а сам `X` на `new.cluster`:

- создать новый профиль PG и переопределить в нём `host/port/connectDB` сущности (одну из них как минимум):

```
$ grep -B 9 -A 1 myfakeprofilename  ./env-config/env-values.yaml
postgres:
  installed: True
  host: 'default.cluster'
  port: 5432
  connectDB: postgres
  storageSizeMB: 5120
  profiles:
    fakeprofilename:
      host: 'new.cluster'
```

```
      port: 5432
      connectDB: postgres
```

- переключить сервис X на нужный профиль через `service-list-patch.yaml`:

```
$ cat ./services/service-list-patch.yaml
modules:
  - name: rtdb
    services:
      - name: zif-rtdb-metadata
        postgresProfile: fakeprofilename
```

- **Внимание**! Если Вы забыли (или намеренно) не указали параметры в профиле они будут взяты из `postgres.host / postgres.port / postgres.connectDB`, т.е.:

```
postgres:
  host: 'default.cluster'
  port: 5555
  connectDB: postgres1
  profiles:
    fakeprofilename:
      connectDB: postgres
```

Обращение выше можно заменить аналогичным более подробным:

```
postgres:
  host: 'default.cluster'
  port: 5555
  connectDB: postgres1
  profiles:
    fakeprofilename:
      host: 'default.cluster'
      port: 5555
      connectDB: postgres
```

- пустой профиль является некорректной конфигурацией, потому как нет смысла создавать пустой профиль в виду пункта выше:

```
# некорректная конфигурация
postgres:
  host: 'default.cluster'
  port: 5555
  connectDB: postgres1
  profiles:
    myprofile1:
    myprofile2:
```

- после создания двух дополнительных кластеров PG для двух баз и не используете никакие балансировщики, то типичная конфигурация будет выглядеть так:

```
postgres:
  host: 'default.cluster'
  port: 5432
  connectDB: postgres1
  profiles:
    cluster1:
      host: 'custom1.cluster'
    cluster2:
      host: 'custom2.cluster'
# port не нужно указывать, если он совпадает с postgres.port
# В данном случае у меня 3 кластера на разных машинах, мастера которых доступны по hostname:
# 1) custom1.cluster
# 2) custom2.cluster
# 3) default.cluster
```

- если установлен балансировщик и несколько кластеров PG, то порядок настройки:
  1) На каждом из кластеров создать маркировочные БД "пустышки" для создания маршрута через балансировщик для `init job`, например `cluster1db` , `cluster2db`, `cluster3db`.
  2) Настройте профили:

```
postgres:
  host: 'pgbouncer.manualinfra'
  port: 5432
  connectDB: cluster1db
  profiles:
    secondcluster:
      connectDB: cluster2db
    thirdcluster:
      connectDB: cluster3db
# host и port в каждом из профилей не требуется дублировать, т.к. он совпадает с postgres.host и postgres.port
# В данном случае у меня 3 кластера на разных машинах, ко всем мы ходим через балансировщик на pgbouncer.manualinfra:5432
```

3) Переключите сервисы на нужный профиль:

```
$ cat ./services/service-list-patch.yaml
modules:
  - name: rtdb
    services:
      - name: zif-rtdb-metadata
        postgresProfile: thirdcluster
  - name: om
    services:
      - name: zif-om-object
        postgresProfile: secondcluster
```

4) Произвести команду `zifctl sync`.

6. Автоматическое создание и конфигурация топиков Kafka в процессе развертывания

Система развертывания должна автоматически создавать топики в `Kafka` с указанными параметрами (`replicas`, `partitions`) и обновлять параметры топиков, если они изменились.

В рамках данной задачи решается вопрос только управления "статическими" топиками, состав и названия которых известны заранее на этапе конфигурации системы. Управление "динамическими" топиками, которые создаются сервисами в процессе работы "по запросу" вне этой задачи.

В рамки этой задачи так же входит управление топиками `Debezium` (`zifctl` должен взять на себя создание и изменение параметров топиков). Конфигурация `Debezium` не меняется относительно существующей, просто добавляется их предварительное создание вместе со всеми остальными.

Изначально заложенные платформой параметры топиков определены в файле `/services/kafka-topics-list.yaml`, по формату структуре аналогичен `/services/service-list-patch.yaml`. Так же файл является статичным в образе `zifctl` и изменению не подлежит.

Начиная с 2.16.0 появилась возможность управления `kafka` топиками.

Для того, чтобы внести изменения в `kafka` топики для конкретного экземпляра Платформы, вам необходимо создать патч-файл `/services/service-list-patch.yaml` в каталоге конфигурации окружения, в котором будут перечислены те модули и сервисы, в параметры которых вы хотите внести корректировки:

- для включения роли по управлению топиков необходимо убедиться что в `env.values.yaml` включена `tasks -kafka` или `-all`. Пример файла `service-list-patch.yaml``:

```
init:
  tasks:
    - all
```

- для управления топика необходимо в `/services/service-list-patch.yaml` на уровне сервиса добавить блок `kafka`: вложенным словарем топиков и их параметрами. Все указанные параметры указаны в примере ниже:

```
modules:
  - name: <<module>>
    services:
      - name: <<service>>
        enabled: true
        kafka:                                          #required.
           enabled: true                                #required.
           topics:                                      #required.
              topic-1:                                        #required. Должно
совпадать с названием топика без префикса "tenantid__"
```

```
                name: "topic-1"                                #required. Наименование
топика      без      префикса      "tenantid__".      Пр.      <<tenantid>>__zif-om-
object__dbz.public.properties_s,     должен     быть     указан     как     zif-om-
object__dbz.public.properties_s

                partitions: "1"                                 #required. default =
"1". ВАЖНО!!! Данный параметр не может быть ниже чем текущее количество партиций в
указанном топике

                cleanup_policy: "delete,compact"               #optional. default =
"delete" ["compact","delete","delete,compact"]

                segment_ms: "604800000"                        #optional. default =
"604800000" (7 days) [1,...]

                flush_messages: "9223372036854775807"          #optional. default =
"9223372036854775807" [1,...]

                max_message_bytes: "1048588"                   #optional. default =
"1048588" [0,...]

                min_insync_replicas: "1"                       #optional. default =
count pods cp-kafka

                retention_bytes: "-1"                         #optional. default = "-
1" [-1, 0,...]

                retention_ms: "86400000"                       #optional. default =
"86400000" (1 day) [-1, 0,...]

                flush_ms: "9223372036854775807"                #optional. default =
"9223372036854775807" [0,...]

                segment_bytes: "1073741824"                    #optional. default =
"1073741824" (1 gibibyte) [14,...]

                segment_jitter_ms: "0"                        #optional. default = "0"
[0,...]

                unclean_leader_election_enable : "false"     #optional. default =
"false" ["false","true"]

              topic-2:

                name: "topic-2"

                partitions: "2"

                retention_ms: "100000"

                flush_ms: "200000"

                cleanup_policy: "compact"

                segment_ms: "2000000000"

              topic-N:

                name: "topic-N"

                partitions: "2"
```

```
  - name: <<module>>
    services:
      - name: <<service>>
        enabled: true
        kafka:
          enabled: true
          topics:
            topic-N42:
              name: "topic-N42"
              partitions: "16"
              retention_ms: "14400200"
```

- для 2.16.0 `/services/kafka-topics-list.yaml` имеет следующий вид:

```
modules:
  - name: om
    services:
      - name: zif-om-object
        kafka:
          enabled: true
          topics:
            zif-om-object__dbz.public.propertyconfigurations_s:
              name: "zif-om-object__dbz.public.propertyconfigurations_s"
              partitions: "1"
            zif-om-object__dbz.public.properties_s:
              name: "zif-om-object__dbz.public.properties_s"
              partitions: "1"
            zif-om-object__dbz.public.objects_s:
              name: "zif-om-object__dbz.public.objects_s"
              partitions: "1"
```

Технические детали решения:

1) В репозитории системы развертывания состав релиза (список топиков и их параметры) хранится в файле `/services/kafka-topics-list.yaml`.

2) При выполнении установки утилита `zifctl` с помощью вспомогательного скрипта `/scripts/hooks/prepare/15-merge-services-list.py` выполняет полный merge `/services/kafka-topics-list.yaml` с `/services/service-list-patch.yaml` (если этот файл существует).

При конфликте значение берется из `/services/service-list-patch.yaml`. Далее полученный файл объединяется с `/services/*service-list.yaml` согласно Правилам объединения базового и патч-файлов.

3) Добавлены кастомные роли (`roles`) (`initrolessrv_kafkatasksmain.yaml`), `module_utils` (`scriptslibansiblemodule_utils`), `modules` (`scriptslibansiblemoduleskafka_topic.py`) для управления `kafka topics`.
4) Вызов роли происходит с помощью существующих `Job` формирующих `initmodule00-init.yaml`.
5) Добавлена новая `init tasks ("kafka")` отвечающая за вызов роли.
6) Вынесена конфигурирование топиков по умолчанию в отдельный файл `services/kafka-topics-list.yaml`. Все кастомные изменения вносятся в `services/service-list-patch.yaml`.
7) В базовый образ `zifctl-base` добавлены компоненты `images/zifctl-base/requirements.txt`:

```
kafka-python>=2.0.0,<2.1.0
kazoo==2.6.1
pure-sasl==0.5.1
jsonmerge==1.9.2
```

8) Управление топиков основано на https://github.com/StephenSorriaux/ansible-kafka-admin.

7. Включение TLS, аутентификации и авторизации в Redis

Для обеспечения безопасности данных и шифрование соединения необходимо реализовать защищенное `TLS` соединение с `Redis` а так же добавить возможность включение аутентификации и авторизации.

В данной документации будет рассмотрено, как включить различные режимы безопасности и аутентификации для `Redis`, а также какие параметры буду добавлены в деплойменты в каждом режиме.

7.1. Включение режима TLS (Transport Layer Security)

`TLS` обеспечивает шифрование данных между клиентами и сервером `Redis` для обеспечения безопасности передачи данных.

Для включения режима `TLS`, установите следующий параметр в файле конфигурации `env-values.yaml`:

```
pki:
  enabled: true
redis:
  tls:
    enabled: true
```

Создаваемые параметры:

При включении режима `TLS` будут сгенерированы следующие переменные окружения:

- `REDIS_HOST=\<адрес подключения к redis>`;
- `REDIS_PORT=\<порт подключения, по умолчанию 6380>`;
- `REDIS_SSL=true`: Устаревшая переменная для указания поддержки `TLS`. Оставлена для совместимости;
- `REDIS_TLS_ENABLED=true`: Эта переменная указывает на поддержку `TLS`;
- `REDIS_TLS_CA_CRT_PATH=\<путь к корневому сертификату>`: Путь к корневому сертификату для `TLS`.

Подключены и смонтированы следующие `volumes`:

- `zif-generic-tls`;
- `zif-redis-tls`.

7.2. Включение режима AUTH (аутентификация)

Режим `AUTH` предоставляет аутентификацию клиентов при подключении к серверу `Redis`.

Для включения режима `AUTH`, установите следующий параметр в файле конфигурации `env-values.yaml`:

```
redis:
  auth: true
```

Создаваемые переменные окружения:

При включении режима `AUTH` будет сгенерирована следующая переменная окружения:

- `REDIS_HOST=\<адрес подключения к redis>;`
- `REDIS_PORT=\<порт подключения, по умолчанию 6379>;`
- `REDIS_PASSWORD=\<ваш пароль>`: Эта переменная указывает на установленный пароль для аутентификации.

7.3. Включение режима AUTH и ACL

Режим `AUTH` и `ACL` объединяет в себе аутентификацию клиентов и управление списками контроля доступа для максимальной безопасности и управления доступом.

Для включения режима `AUTH` и `ACL`, установите следующие параметры в файле конфигурации `env-values.yaml`:

```
redis:
  auth: true
  acl: true
```

Создаваемые переменные окружения при включении режима `AUTH` и `ACL` будут сгенерированы следующие переменные окружения:

- `REDIS_HOST=\<адрес подключения к redis>;`
- `REDIS_PORT=\<порт подключения, по умолчанию 6379>;`
- `REDIS_PASSWORD=\<ваш пароль>`: Эта переменная указывает на установленный пароль для аутентификации;
- `REDIS_USER=\<имя пользователя, по умолчанию tenant_modulename>`.

7.4. Включение режима TLS, AUTH и ACL

Режим `TLS` и `ACL` объединяет в себе шифрование данных, обязательной аутентификации и управление списками контроля доступа для максимальной безопасности.

Для включения режима `TLS` и `ACL`, установите следующие параметры в файле конфигурации `env-values.yaml`:

```
pki:
  enabled: true
redis:
```

```
  tls:
    enabled: true
  auth: true
  acl: true
```

При включении режима `TLS` и `ACL` должны создаваться все переменные окружения и монтироваться `volumes` как и в режимах `TLS` и `ACL`.

7.5. Режим публикации внешнего Ingress/Route соединения для сервера Redis

Этот режим позволяет обеспечить внешний доступ к серверу `Redis` через `Ingress` или `Route` с использованием `SSL passthrough`.

Параметры, необходимые для включения:

```
pki:
  enabled: true
redis:
  tls:
    enabled: true
  externalAccess: True
```

Для корректного создания внешнего доступа к `redis` в кластере с `ingress nginx` должен быть включен параметр `--enable-ssl-passthrough` при запуске `ingress controller`.

Создаваемые сущности:

При включении внешнего доступа будет создан `ingress/route` с адресом формата `\<namespace>-redis.\<domain.example>`.

В ingress присутствует аннотация `[nginx.ingress.kubernetes.io/ssl-passthrough](http://nginx.ingress.kubernetes.io/ssl-passthrough): "true"\`, указывающая на использовании `nginx` режима `ssl-passthrough` для этого адреса.

Подключение:

При подключении через любую утилиту или программу нужно указать:

- адрес указанный в `ingress/route`;
- порт подключения 443;
- указать путь до `CA` сертификата или сертификат в текстовом виде (зависит от способа);
- указать `SNI` который равен адресу;
- при использовании авторизации указать логин и пароль.

8. Включение отдельного Redis для инфраструктуры и настройка Apache Nifi

В ходе тестирования работы сервисов `UDL` обнаружилась избыточная сетевая нагрузка на сервис `zif-redis-master`, вызванная логикой работы платформенного процессора `Apache Nifi\_ProcessTSDSData`.

Для решения этой проблемы в авто-развертывание добавлена возможность установки отдельного экземпляра `Redis`, который будет использоваться только инфраструктурными компонентами платформы.

В документе описаны действия, необходимые для установки инфраструктурного `Redis` и конфигурации процессоров `Nifi`, использующих `Redis`.

8.1. Установка Redis для инфраструктуры

При переустановке на уже развернутую Платформу:

1) Добавьте в файл конфигурации `*\<путь до папки конфигурации zifctl>/env-config/env-values.yaml*` следующие параметры (включение `Redis` для инфраструктурных сервисов):

```
env-values.yaml | Включение Redis для инфраструктурных сервисов
## Конфигурация Redis используемого только инфраструктурными сервисами
redisInfra:
  installed: True
  host: 'zif-redis-infra-master.<namespace>'
  auth: True
  port: 6379 # по умолчанию, можно не указывать
  storageSizeMB: 1024 # по умолчанию, можно не указывать
```

2) Расшифровуйте файл в секретами `*\<путь до папки конфигурации zifctl>/env-config/env-secrets.yaml*`, используя команду `zifctl secrets dec` бинарного установщика `zifctl` (Дешифрование):

```
env-secrets.yaml | Дешифрование
./zifctl secrets dec --env-path=$ENV_PATH --age-key=$AGE_KEY
 2023-09-05 16:15:25,083 INFO    secrets: Write decrypted file: /var/deploy/env/env-secrets.dec.yaml, don't forget to delete it!
```

В результате будет создан файл `*\<путь до папки конфигурации zifctl>/env-config/env-secrets.dec.yaml*`, в который необходимо добавить параметры для `redisInfra` (Добавление параметров `Redis` для инфраструктуры):

```
env-secrets.yaml | Добавление параметров Redis для инфраструктуры
...
redisInfra:
    adminPassword: <password>
    adminUser: admin
...
```

3) Зашифруйте файл при помощи команды `zifctl secrets enc`:

```
env-secrets.yaml | Шифрование
./zifctl secrets enc --env-path=$ENV_PATH --age-key=$AGE_KEY
 2023-09-05 16:24:55,527 INFO   secrets: Encrypt secrets file: /var/deploy/env/env-secrets.dec.yaml => /var/deploy/env/env-secrets.yaml
 2023-09-05 16:24:55,539 INFO   secret_store_base: Save secrets to file: /var/deploy/env/env-secrets.yaml
```

В результате в \<путь до папки конфигурации zifctl>/env-config/env-secrets.yaml`` будут добавлены зашифрованные параметры redisInfra`.

После удаления файла *\<путь до папки конфигурации zifctl>/env-config/env-secrets.dec.yaml* можно начать установку платформы командой `zifctl sync`.

При установке с нуля:

В `env-values.yaml` и `env-secrets.yaml` уже будут записаны параметры для `redisInfra`, перед выполнением команды `zifctl sync` необходимо только включить установку `redisInfra` (с аутентификацией):

```
env-values.yaml | Включение Redis для инфраструктурных сервисов
...
## Конфигурация Redis используемого только инфраструктурными сервисами
redisInfra:
  installed: True
  auth: True
...
```

8.2. Конфигурация Apache Nifi

Для тестирования корректной работы `Nifi` с `Redis` необходимо воспользоваться руководством из данного раздела.

Добавление сервисов контроллера `RedisConnectionPoolService` и `RedisDistributedMapCacheClientService`:

1) Войти в UI `Nifi` под административной УЗ.
2) Нажмите ПКМ и выберите `Configure`:

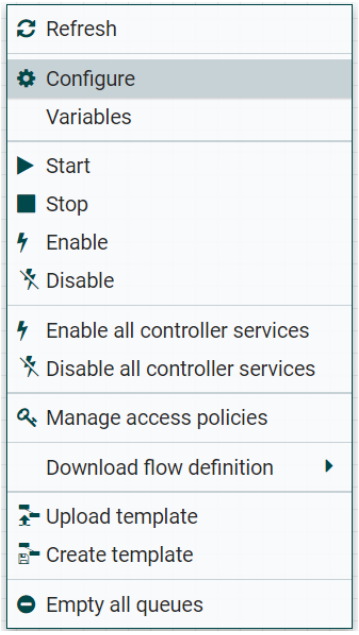

Рисунок 8.1 Configure

3) Перейдите на вкладку `Controller Services → кнопка "+"` в правом верхнем углу.
4) В открывшемся диалоговом окне `Add Controller Service` в текстовом поле `Filter` введите `Redis`:

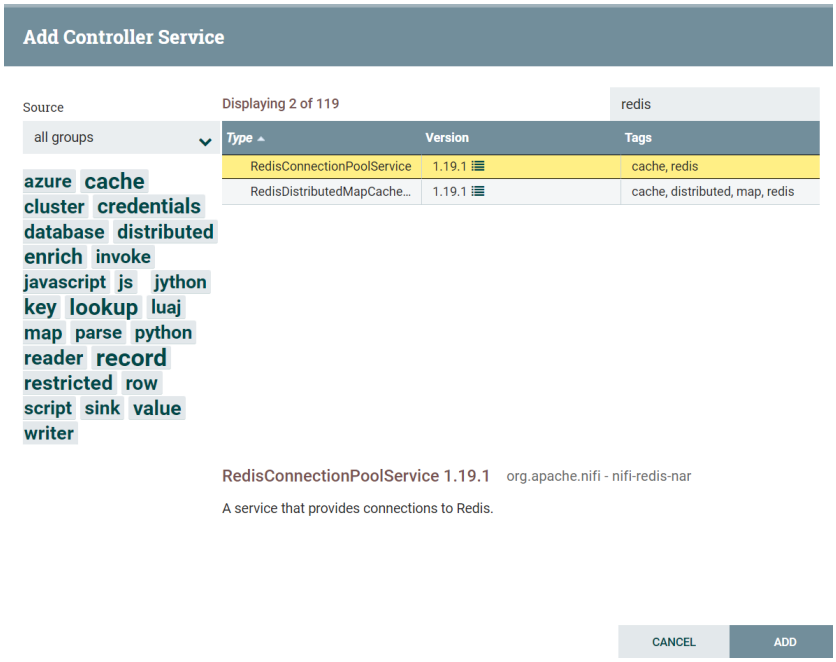

Рисунок 8.2 Redis

Последовательно добавляем сервисы контроллера `RedisConnectionPoolService`и `RedisDistributedMapCacheClientService`.

8.3. Конфигурация сервисов контроллера RedisConnectionPoolService и RedisDistributedMapCacheClientService

Конфигурация сервиса контроллера `RedisConnectionPoolService`:

1) Выберите кнопку конфигурации `RedisConnectionPoolService`:

Рисунок 8.3 RedisConnectionPoolService

2) В открывшемся диалоговом окне `Configure Controller Service` `RedisConnectionPoolService 1.19.1`, вкладка `Properties`, введите значения полей:

   - `Connection String`: `zif-redis-infra-headless.<namespace>:6379`;
   - `Password`: пароль администратора `redisInfra`; (`redisInfra.adminPassword` из `env-secrets.yaml` либо значение переменной `REDIS_INFRA_ADMIN_PASSWORD` из секрета `zif-global-secrets` в кластере);
   - `Apply`.

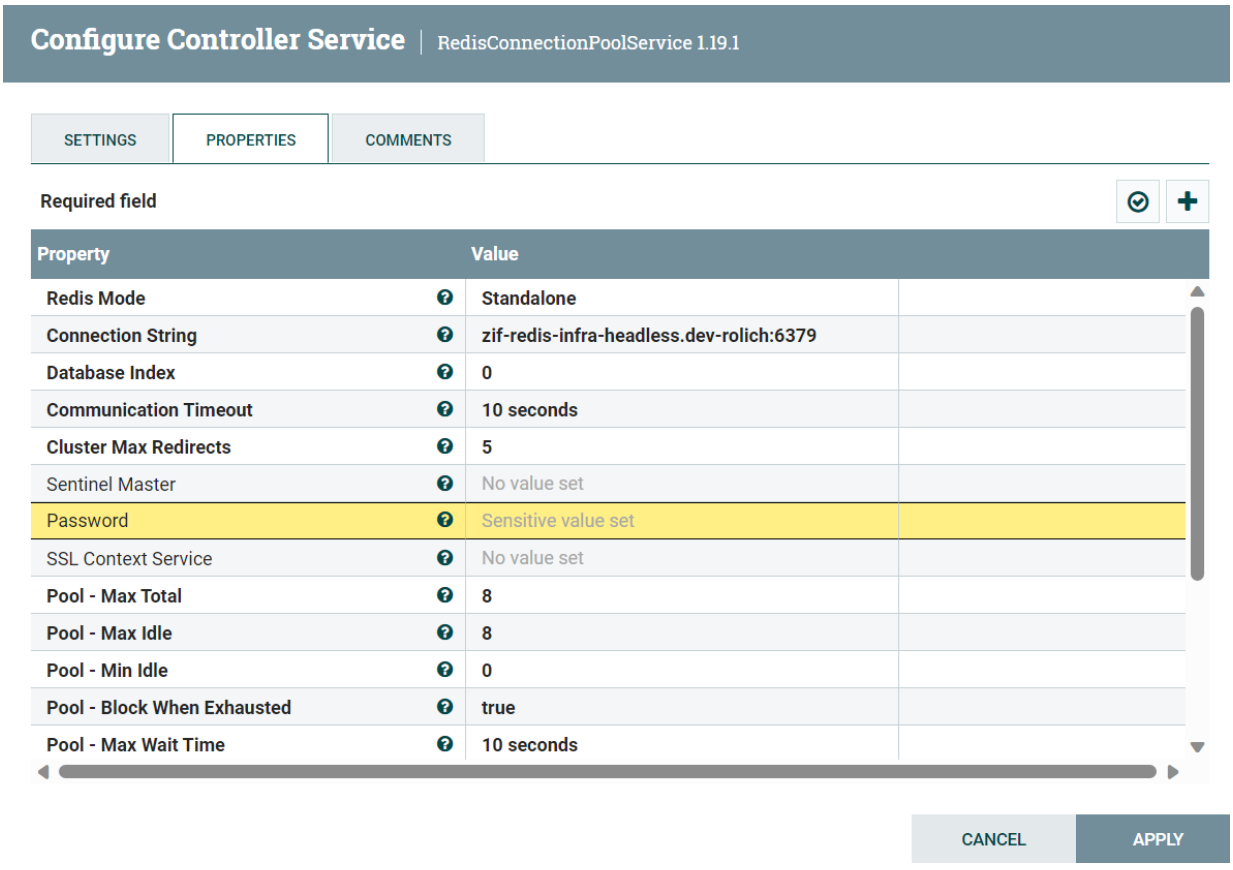

Рисунок 8.4 Вкладка Properties

Конфигурация сервиса контроллера `RedisDistributedMapCacheClientService`:

1) Нажмите кнопку конфигурации сервиса `RedisDistributedMapCacheClientService`.
2) В открывшемся диалоговом окне `Configure Controller \| ServiceRedisDistributedMapCacheClientService 1.19.1` выберите вкладку `Properties`.
3) Поле `Redis Connection Pool``, нажмите` Value, `в выпадающем списке выберите` RedisConnectionPoolService → OK → Apply`.

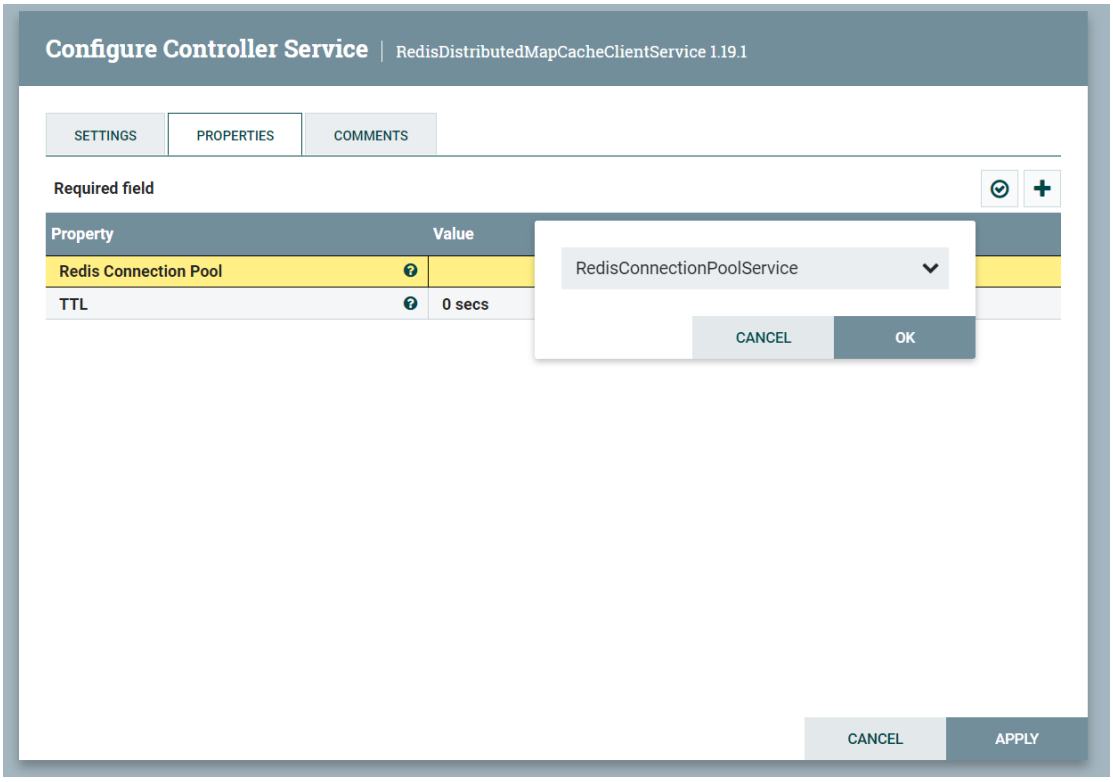

Рисунок 8.5 Configure Controller

8.4. Включение сервисов контроллера RedisConnectionPoolService и RedisDistributedMapCacheClientService

1) Включите сервис `RedisConnectionPoolService`:

Рисунок 8.6 RedisConnectionPoolService

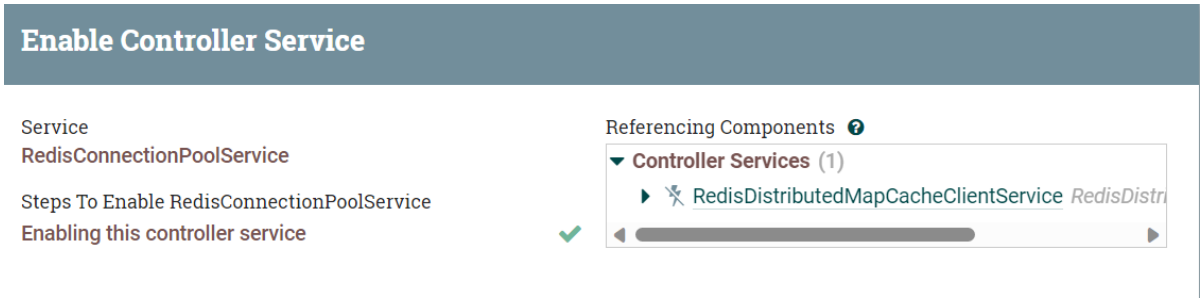

Рисунок 8.7 RedisConnectionPoolService

2) Включите сервис `RedisDistributedMapCacheClientService`:

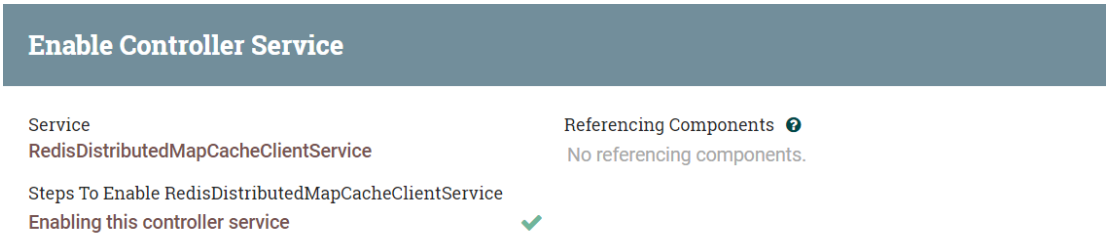

Рисунок 8.8 RedisDistributedMapCacheClientService

Добавление процессоров `GenerateFlowFile` и `PutDistributedMapCache`:

1) Для добавления процессоров наведите курсор на `Processors` и перетащите на поле:

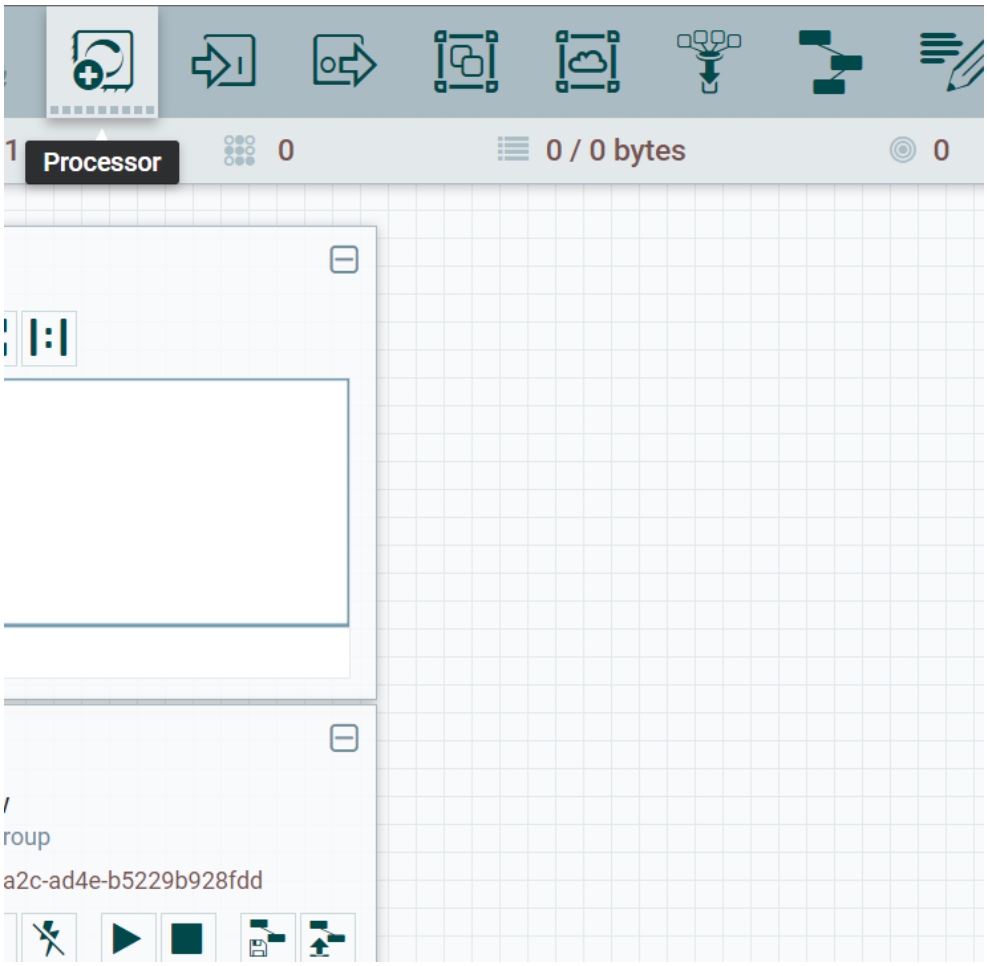

Рисунок 8.9 Processors

2) В открывшемся диалоговом окне `Add Processor` в поле `Filter` введите `GenerateFlowFile` → `Add`:

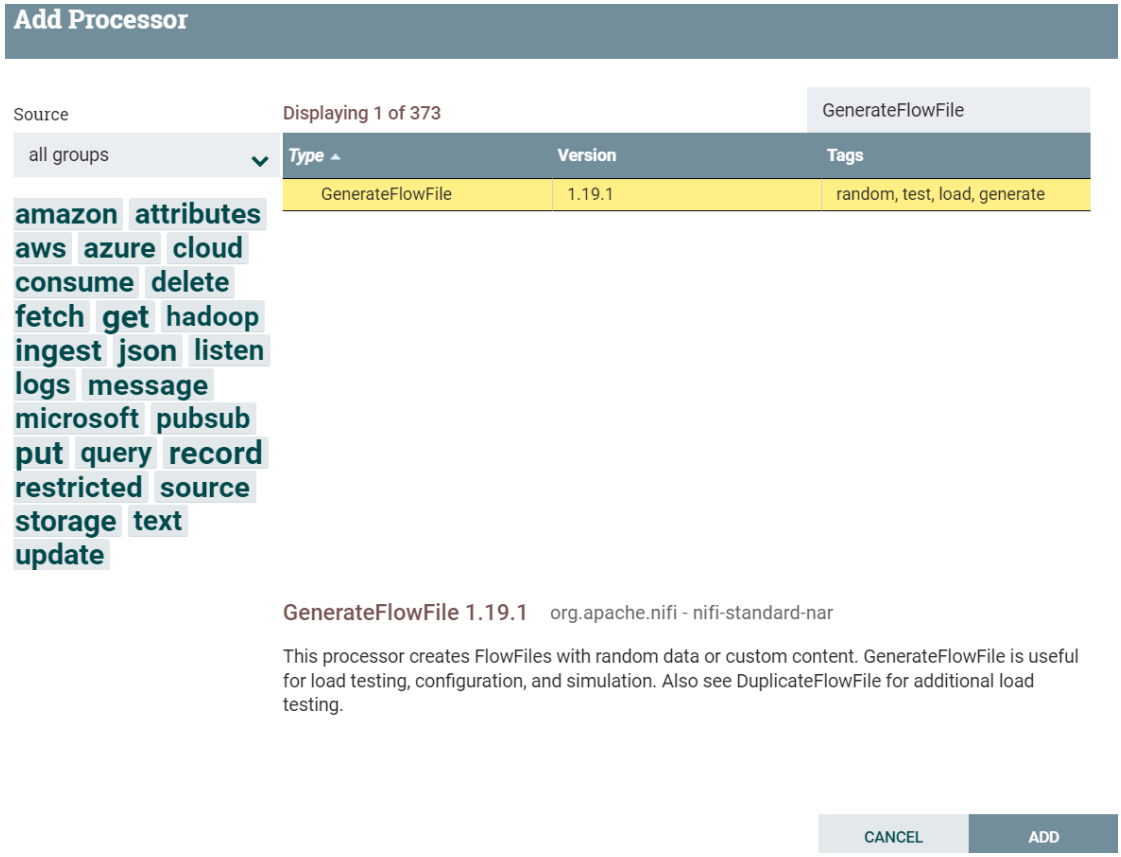

Рисунок 8.10 GenerateFlowFile → Add

3) Аналогичным образом добавьте процессор `PutDistributedMapCache`:

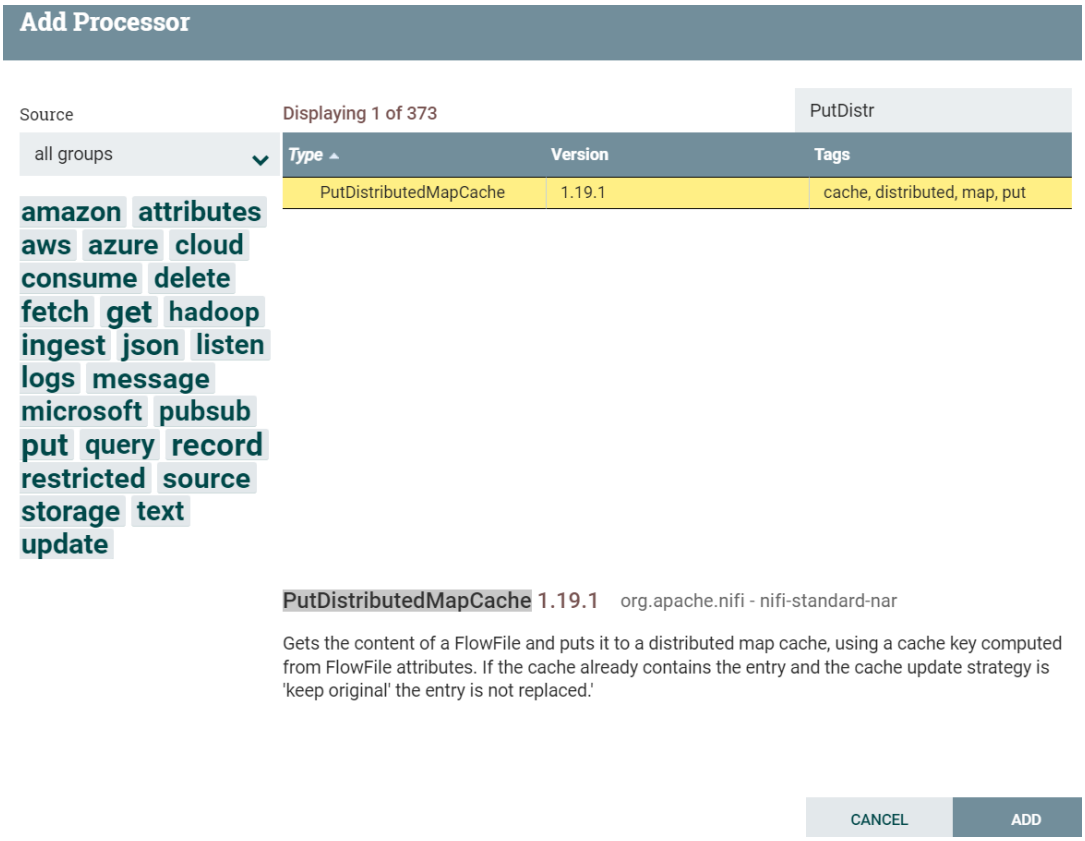

Рисунок 8.11 PutDistributedMapCache

4) Нажмите на стрелочку процессора `GenerateFlowFile` и тянем до `PutDistributedMapCache`:

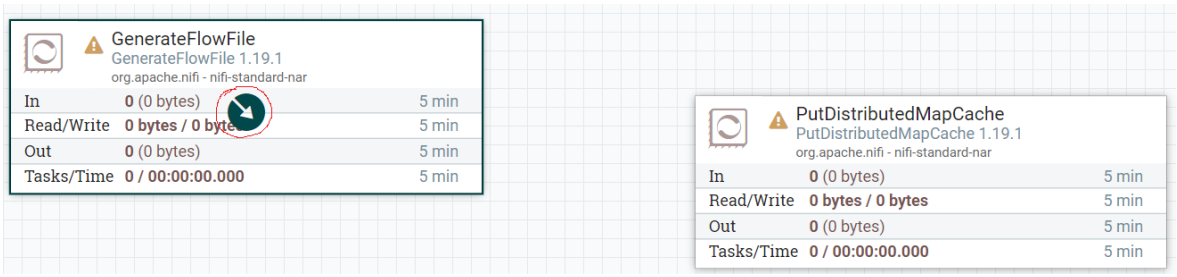

Рисунок 8.12 PutDistributedMapCache

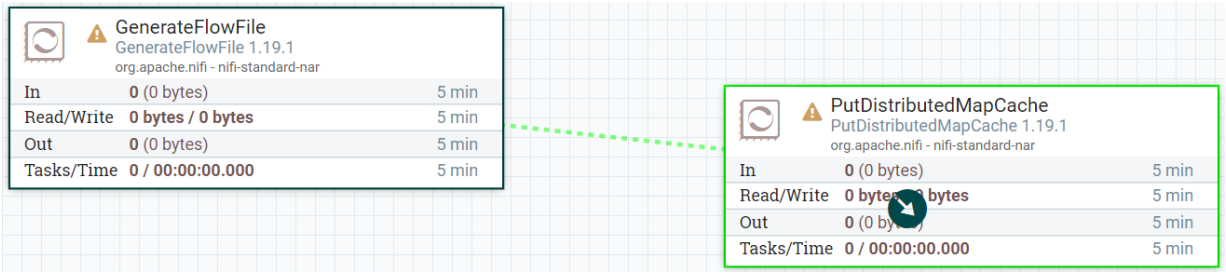

Рисунок 8.13 PutDistributedMapCache

5) В открывшемся диалоговом окне выберите `Create Connection → Add`.

8.5. Конфигурация процессоров GenerateFlowFile и PutDistributedMapCache

Конфигурация процессора `GenerateFlowFile`:

1) Двойной клик (либо ПКМ), затем `Configure` на процессоре `GenerateFlowFile`.
2) Вкладка `Properties`, в поле `Custom Text` введите `hello from nifi`, нажмите `Apply`:

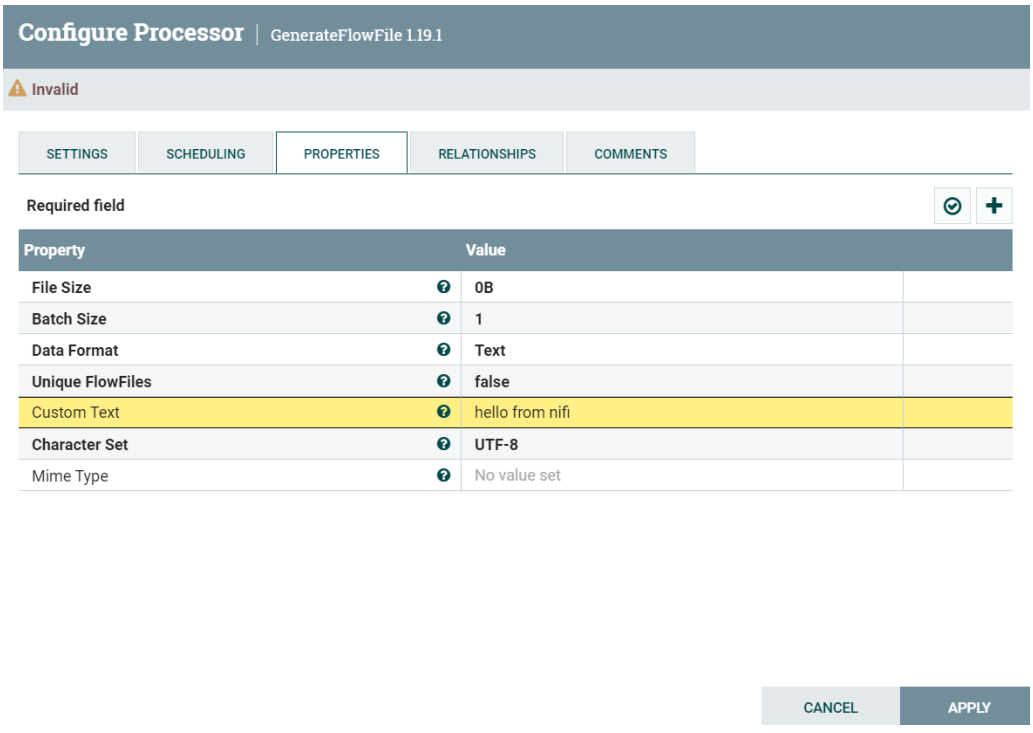

Рисунок 8.14 Properties

Конфигурация процессора `PutDistributedMapCache`:

1) Двойной клик (либо правый клик → `Configure`) на процессоре `PutDistributedMapCache`.
2) Вкладка `Properties`, в значения в следующие поля:
   - **Cache Entry Identifier**: nifi-key;
   - **Distributed Cache Service**: из выпадающего списка выбираем **RedisDistributedMapCacheClientService.

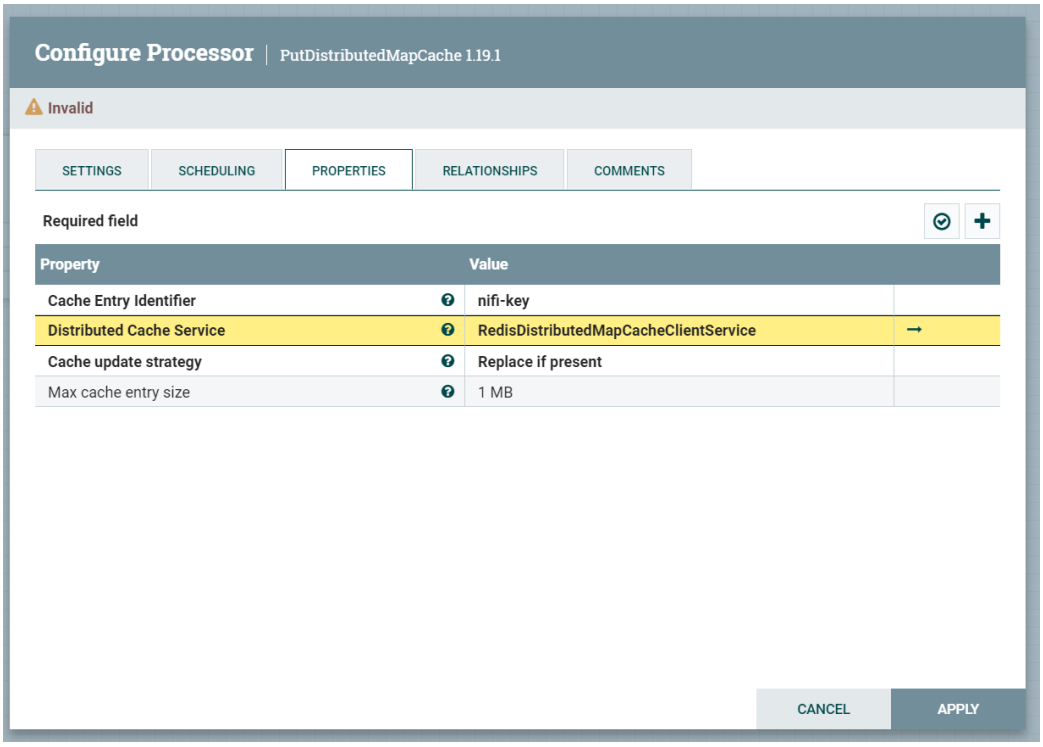

Рисунок 8.15 Properties

3) Вкладка `Relationships`, выберите галочки `terminate` в секциях `failure` и `success`:

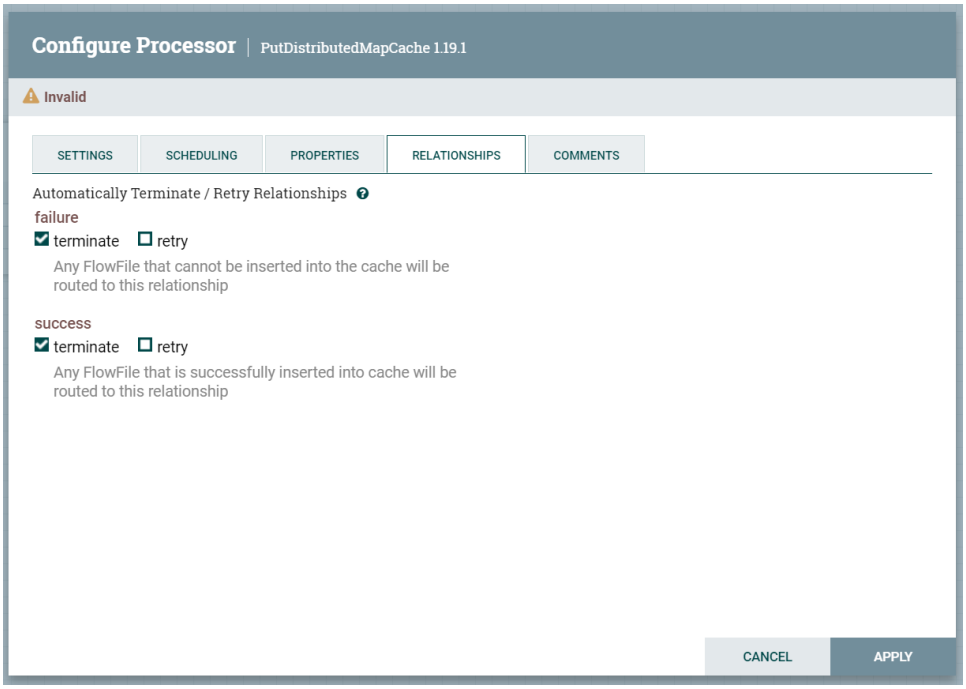

Рисунок 8.16 Relationships

8.6. Запуск процессоров GenerateFlowFile и PutDistributedMapCache

1) Нажмите ПКМ на процессоре `PutDistributedMapCache`, нажмите `Start`.
2) Нажмите ПКМ на процессоре `GenerateFlowFile`, затем `Run Once`.
3) Нажмите ПКМ на самом поле (вне процессоров), затем `Refresh`.
4) Убедитесь про отсутствие ошибок в правом верхнем угле процессора `PutDistributedMapCache`:

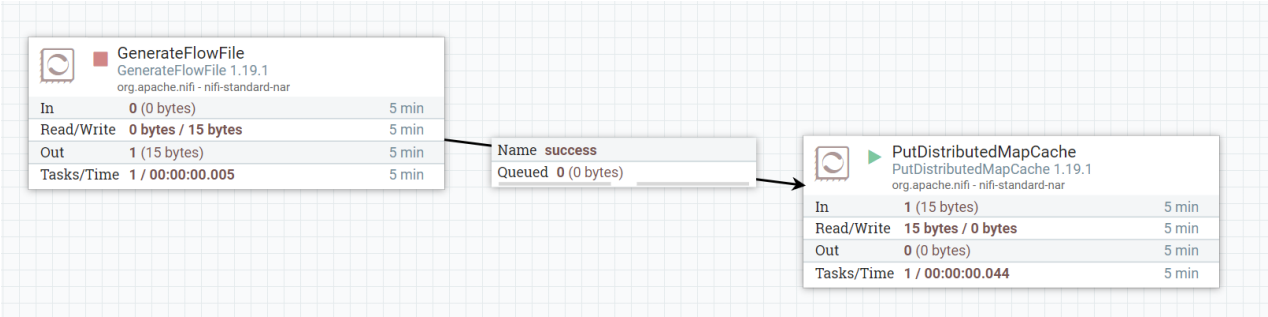

Рисунок 8.17 Relationships

8.7. Тестирование Redis

1) Используйте `Lens` заходим в оболочку (`shell`) пода `zif-redis-infra-master-0.
2) Введите команды:

```
**redis-cli
AUTH "\<пароль администратора Redis>"
KEYS \* →** в результате должен быть выведен ключ **nifi-key
GET nifi-key →** значение ключа должно быть **"hello from nifi"
```

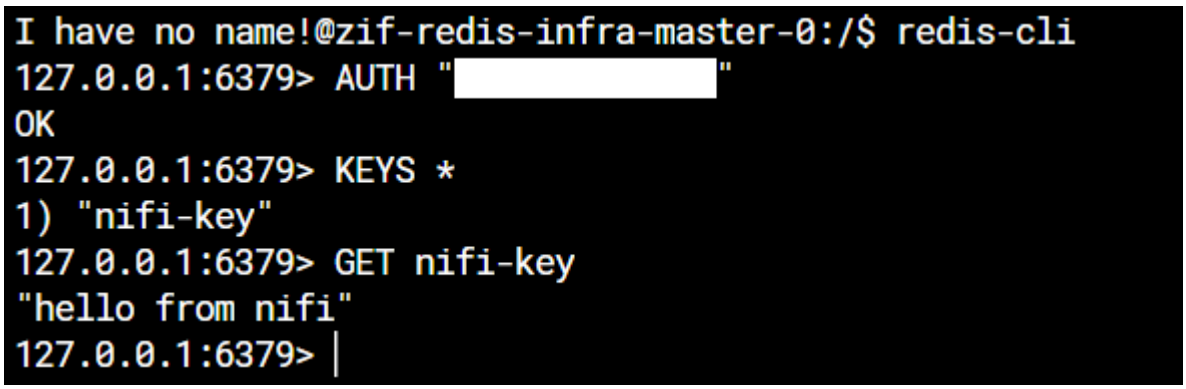

Рисунок 8.18 Тестирование Redis

9. Требования к рабочей станции, с которой будет производиться развертывание, и ее окружению

Таблица 9.1. Требования к оборудованию/рабочей станции

	Минимальные	Рекомендуемые
ЦПУ	2	4
ОЗУ	4 Гб	8Гб

Требования к окружению:

- ОС семейства **Linux** или **wsl2** (для ОС **Windows**).
- **Docker**.
- **Python** 3.10.

Для развертывания рекомендуется использовать скрипт-обертку (**wrapper script**).

Запуск системы автоматического развертывания платформы (**ZIIoT Deployment system** или **zifctl**) можно осуществить двумя способами:

- Выполнение развертывания с помощью **wrapper**, запуск **zifctl** интерактивно (рекомендуемый способ запуска).
- Запуск **zifctl** из контейнера **docker**.